# COMPUTER GRAPHICS INFR 2350 GDW REPORT

ROBERT ANDERSEN - 100658473

LILLIAN FAN - 100672027

ROWAN LUCKHURST - 100657644

SAFA NAZIR - 100654328

JOSHUA SANKARLAL - 100658457

EMORY WYNN - 100655604

# RENDERING PROCESS

First we stores position, normal, & colour data to separate textures in m_mainFrameBuffer (excluding transparent objects). These are all used later down the line when doing lighting calculations.

```
//updating viewport to fit framebuffer size
glViewport(0, 0, getWindowWidth(), getWindowHeight());
m_mainFrameBuffer->enable();
m_mainCamera->render(m_modelShader, m_models, false);
m_mainFrameBuffer->disable();
```

We then take information stored in the m_mainFrameBuffer and apply the lighting calculations and stores it in m_postBuffer.

```
///~ store data for post process ~///

m_postBuffer->enable();
m_postProcess->enable();

//bind textures
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, m_mainFrameBuffer->getColorHandle(0));
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, m_mainFrameBuffer->getColorHandle(1));
glActiveTexture(GL_TEXTURE2);
glBindTexture(GL_TEXTURE_2D, m_mainFrameBuffer->getColorHandle(2));
glActiveTexture(GL_TEXTURE3);
glBindTexture(GL_TEXTURE_2D, tmpRamp.id);

m_postProcess->sendUniform("uPos", 0);
m_postProcess->sendUniform("uNorm", 1);
m_postProcess->sendUniform("uScene", 2);
m_postProcess->sendUniform("uRamp", 3);
m_postProcess->sendUniform("toonActive", toonActive);

FrameBuffer::drawFullScreenQuad();
```

```
//un-bind textures
glActiveTexture(GL_TEXTURE3);
glBindTexture(GL_TEXTURE_2D, GL_NONE);
glActiveTexture(GL_TEXTURE2);
glBindTexture(GL_TEXTURE_2D, GL_NONE);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, GL_NONE);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, GL_NONE);

m_postProcess->disable();
m_postBuffer->disable();
```

After all the opaque objects are rendered, we then Draw transparent objects to the m_postBuffer.

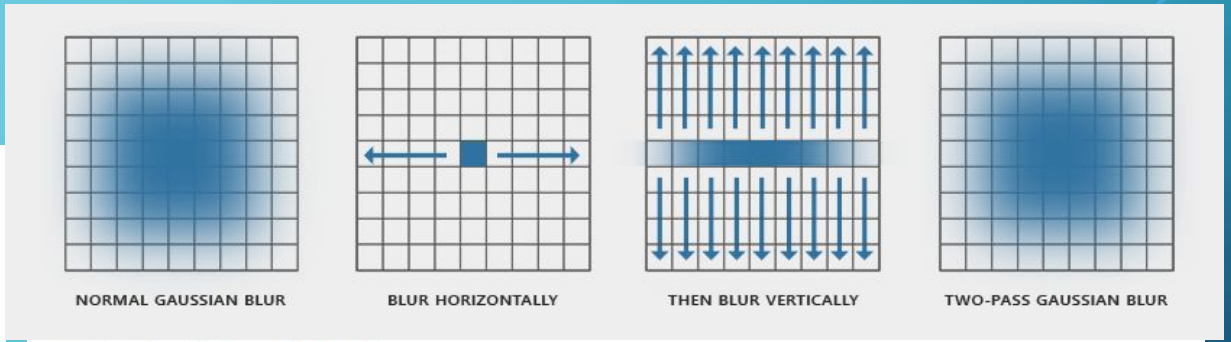
```
m_mainFrameBuffer->moveDepthToBuffer(getWindowWidth(), getWindowHeight(),
m_postBuffer->getFrameBufferID());

// render transparent objects to the framebuffer
m_postBuffer->enable();

LightSource::setShader(m_forwardRender);
LightSource::update();
m_mainCamera->render(m_forwardRender, m_models, true);

m_postBuffer->disable();
```

# BLOOM

The first effect applied as a post processing effect is bloom. We first need to down scale the buffer and get the high pass for bloom which takes the brightest colours from the scene based on a threshold and disregards other colours.

```
 ///~ Bloom ~///
// update the viewport to fit framebuffer size
glViewport(0, 0, getWindowWidth() / SCREEN_RATIO, getWindowHeight() /
SCREEN_RATIO);

//binds the initial bloom affect to buffer 1
m_buffer1->enable();
m_bloomHighPass->enable();

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, m_postBuffer->getColorHandle(0));

m_bloomHighPass->sendUniform("uTex", 0);
m_bloomHighPass->sendUniform("uThresh", 0.15f);

FrameBuffer::drawFullScreenQuad();

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, GL_NONE);

m_bloomHighPass->disable();
m_buffer1->disable();
```

We loop through two buffers where one takes the high pass and applies Gaussian Blur on the horizontal and vertical axis separately




```
//Takes the high pass and blurs it
for (int a = 0; a < SCREEN_RATIO; a++)
{
 m_buffer2->enable();
 m_blurHorizontal->enable();
 m_blurHorizontal->sendUniform("uTex", 0);
 m_blurHorizontal->sendUniform("uPixleSize", 1.0f / getWindowHeight());
 glBindTexture(GL_TEXTURE_2D, m_buffer1->getColorHandle(0));
 FrameBuffer::drawFullScreenQuad();

 glBindTexture(GL_TEXTURE_2D, GL_NONE);
 m_blurHorizontal->disable();
 m_buffer1->enable();
 m_blurVertical->enable();
 m_blurVertical->sendUniform("uTex", 0);
 m_blurVertical->sendUniform("uPixleSize", 1.0f / getWindowWidth());
 glBindTexture(GL_TEXTURE_2D, m_buffer2->getColorHandle(0));
 FrameBuffer::drawFullScreenQuad();

 glBindTexture(GL_TEXTURE_2D, GL_NONE);
 m_blurVertical->disable();
}
FrameBuffer::disable(); //return to base frame buffer
```

# Below are the Vertical and Horizontal frag shader passes for our bloom effect

```
// horizontal
void main()
{
   outColour.rgb = vec3(0,0,0);
   outColour.a = 1.0;
   outColour.rgb += texture(uTex,vec2(texcoord.x,texcoord.y - 3 *
uPixleSize)).rgb * 0.09;
   outColour.rgb += texture(uTex,vec2(texcoord.x,texcoord.y - 2 *
uPixleSize)).rgb * 0.12;
   outColour.rgb += texture(uTex,vec2(texcoord.x,    texcoord.y -
uPixleSize)).rgb * 0.15;
   outColour.rgb += texture(uTex,vec2(texcoord.x,
texcoord.y)).rgb * 0.16;
   outColour.rgb += texture(uTex,vec2(texcoord.x,    texcoord.y +
uPixleSize)).rgb * 0.15;
   outColour.rgb += texture(uTex,vec2(texcoord.x,texcoord.y + 2 *
uPixleSize)).rgb * 0.12;
   outColour.rgb += texture(uTex,vec2(texcoord.x,texcoord.y + 3 *
uPixleSize)).rgb * 0.09;
}
```

```
// vertical
void main()
{

   outColour.rgb = vec3(0,0,0);
   outColour.a = 1;

   outColour.rgb += texture(uTex,vec2(texcoord.x - 3 *
uPixleSize,texcoord.y)).rgb * 0.09;
   outColour.rgb += texture(uTex,vec2(texcoord.x - 2 *
uPixleSize,texcoord.y)).rgb * 0.12;
   outColour.rgb += texture(uTex,vec2(texcoord.x - uPixleSize,
texcoord.y)).rgb * 0.15;
   outColour.rgb += texture(uTex,vec2(texcoord.x,
texcoord.y)).rgb * 0.16;
   outColour.rgb += texture(uTex,vec2(texcoord.x + uPixleSize,
texcoord.y)).rgb * 0.15;
   outColour.rgb += texture(uTex,vec2(texcoord.x + 2 *
uPixleSize,texcoord.y)).rgb * 0.12;
   outColour.rgb += texture(uTex,vec2(texcoord.x + 3 *
uPixleSize,texcoord.y)).rgb * 0.09;
}
```

After blurring the high pass we then create a composites of the final colour of the scene from the information stored in both m_postBuffer and the blurred high pass.

```
// update the viewport to fit framebuffer size
glViewport(0, 0, getWindowWidth(), getWindowHeight());

//composes both the base scene and the bloom calculations
m_greyscaleBuffer->enable();
m_blurrComposite->enable();

glActiveTexture(GL_TEXTURE0);
m_blurrComposite->sendUniform("uScene", 0);
glBindTexture(GL_TEXTURE_2D, m_postBuffer->getColorHandle(0));

glActiveTexture(GL_TEXTURE1);
m_blurrComposite->sendUniform("uBloom", 1);
glBindTexture(GL_TEXTURE_2D, m_buffer1->getColorHandle(0));
FrameBuffer::drawFullScreenQuad();

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, GL_NONE);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, GL_NONE);

m_blurrComposite->disable();
m_greyscaleBuffer->disable();
```

Lastly we apply grayscale to the scene along with applying the colour correction

```
m_grayScalePost->enable();

m_grayScalePost->sendUniform("uTex", 0);
m_grayScalePost->sendUniform("customTexure", 6);
m_grayScalePost->sendUniform("lutSize", tmpLUT.lutSize);
m_grayScalePost->sendUniform("lutActive", lutActive);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, m_greyscaleBuffer->getColorHandle(0));
glActiveTexture(GL_TEXTURE6);
glBindTexture(GL_TEXTURE_3D,
ResourceManager::getTexture3D(LUTpath.c_str()).id);

FrameBuffer::drawFullScreenQuad();

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, GL_NONE);
glActiveTexture(GL_TEXTURE6);
glBindTexture(GL_TEXTURE_3D, GL_NONE);

m_grayScalePost->disable();
```

# TOON SHADING

In our game we have two kinds of light, one is directional light and another is point light. We applied the toon-shading on the specular component of our point light.

At the start of the process we load the 1D texture ramp wich will be applied as the toon-shading of our light.


```
Texture2D tmpRamp;
```

```
tmpRamp = ResourceManager::getTexture2D("Texture/pinkRamp.png");
```

First, we create a 2D texture and then load the .png into memory.

```
//bind textures
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, m_mainFrameBuffer->getColorHandle(0));
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, m_mainFrameBuffer->getColorHandle(1));
glActiveTexture(GL_TEXTURE2);
glBindTexture(GL_TEXTURE_2D, m_mainFrameBuffer->getColorHandle(2));
glActiveTexture(GL_TEXTURE3);
glBindTexture(GL_TEXTURE_2D,tmpRamp.id);

m_postProcess->sendUniform("uPos", 0);
m_postProcess->sendUniform("uNorm", 1);
m_postProcess->sendUniform("uScene", 2);
m_postProcess->sendUniform("uRamp", 3);
m_postProcess->sendUniform("toonActive", toonActive);

FrameBuffer::drawFullScreenQuad();

//un-bind textures
glActiveTexture(GL_TEXTURE3);
glBindTexture(GL_TEXTURE_2D, GL_NONE);
glActiveTexture(GL_TEXTURE2);
glBindTexture(GL_TEXTURE_2D, GL_NONE);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, GL_NONE);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, GL_NONE);
```

Then we bind this 2D texture as “uRamp” to our post process shader at slot 3 and unbind it afterward to free the slot.

```
uniform sampler2D uRamp;
```

```
void pointLight(int a)
{
    vec3 colour = texture(uScene, texcoord).rgb;
    vec3 normal = normalize(texture(uNorm,texcoord).rgb);
    vec3 lightVec = LightPosition[a].xyz - texture(uPos,texcoord).xyz;
    float dist = length(lightVec);
    vec3 direc = lightVec / dist;
    vec3 reflection = reflect(-direc,normal);
    vec3 eye = normalize(- texture(uPos,texcoord).xyz);
    float viewToRe =  max(dot(eye,reflection),0.0);
    float NdotL = max(dot(normal, lightVec),0.0);

    //The light contributes to this surface
    //Calculate attenuation (falloff)
    float attenuation = 1.0 / (Attenuation_Constant[a] + (Attenuation_Linear[a] * dist)
                               + (Attenuation_Quadratic[a] * dist * dist));

    //Calculate diffuse contribution
    outColor.rgb +=  max(NdotL, 0.0) *  LightDiffuse[a] * attenuation * colour;

    //Calculate specular contribution
    if(toonActive)
    {
        outColor.rgb += LightSpecular[a] * texture(uRamp, vec2(viewToRe, 0.5)).rgb * attenuation;
    }
    else
    {
        float spec = pow(max(viewToRe, 0.0), 16);
        outColor.rgb +=  LightSpecular[a] * spec * attenuation * colour;
    }
}
```

Inside the fragment shader, we first set the texture to a uniform sampler2D. Then inside the point light calculation, at the specular contribution part, instead of calculating specular normally we use view to reflection to find a color value on our 2D texture ramp. Then we use this color as specular to multiply it with specular strength and attenuation. We also implemented a boolean to toggle toon shading on and off when “T” is pressed in game.

Before

After

# COLOR GRADING

We enable Color Grading by applying a 3D lookup table to reference every colour in each pixel on our fullscreen quad. This acts as a filter to change whole screen's color style. First we need load the 3D LUT (i.e. *.cube) as a 3D texture and also record it size for calculation. On the right side is our 3D texture loader. The first half of our loader is read through the .cube file by certain path, then save the information to our 3D texture. The second half is we generate the 3D texture and set 3D texture parameters to what we need in opengl.

```
Texture3D ImageLoader::loadImage3D(const char * LUTpath)
{
    Texture3D texture;
    std::vector<Coord3D> LUT{};

    std::ifstream LUTfile2(LUTpath);

    while(!LUTfile2.eof())
    {
        std::string LUTline;
        getline(LUTfile2, LUTline);
        if(LUTline.empty()) continue;
        if(strstr(LUTline.c_str(), "LUT_3D_SIZE"))
        {
            sscanf_s(LUTline.c_str(), "LUT_3D_SIZE %d", &texture.lutSize);
        }
        float r, g, b;
        if(sscanf_s(LUTline.c_str(), "%f %f %f", &r, &g, &b) == 3) LUT.push_back({r,g,b});
    }

    glEnable(GL_TEXTURE_3D);

    glGenTextures(1, &texture.id);
    glBindTexture(GL_TEXTURE_3D, texture.id);

    glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_REPEAT);

    glTexImage3D(GL_TEXTURE_3D, 0, GL_RGB, texture.lutSize, texture.lutSize,
                 texture.lutSize, 0, GL_RGB, GL_FLOAT, LUT.data());

    glBindTexture(GL_TEXTURE_3D, 0);
    glDisable(GL_TEXTURE_3D);

    LUT.clear();
    return texture;
}
```

```
Texture3D tmpLUT;
tmpLUT = ResourceManager::getTexture3D("Texture/IWLTBAP_Aspen_-_Standard.cube");
```

```
m_grayScalePost->enable();

m_grayScalePost->sendUniform("uTex", 0);
m_grayScalePost->sendUniform("customTexure", 6);
m_grayScalePost->sendUniform("lutSize", tmpLUT.lutSize);
m_grayScalePost->sendUniform("lutActive", lutActive);


glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, m_greyscaleBuffer->getColorHandle(0));
glActiveTexture(GL_TEXTURE6);
glBindTexture(GL_TEXTURE_3D, tmpLUT.id);

FrameBuffer::drawFullScreenQuad();

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, GL_NONE);
glActiveTexture(GL_TEXTURE6);
glBindTexture(GL_TEXTURE_3D, GL_NONE);

m_grayScalePost->disable();
```

We first create the 3D texture and load our .cube file in it. Then we send this 3D texture and also the LUT size in to our grayScalePost shader.

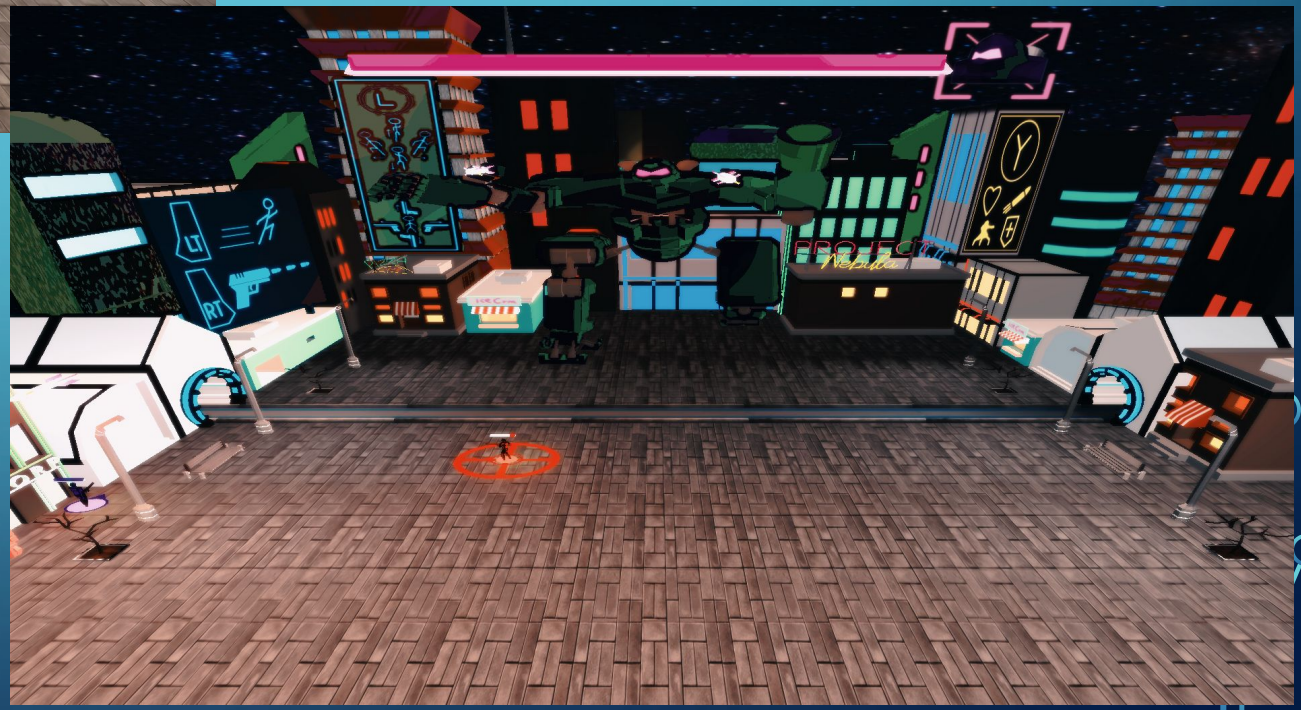
```
layout(binding = 6) uniform sampler3D customTexure;
uniform int lutSize;
uniform bool lutActive;
```

```
void main()
{
    vec4 source  = texture(uTex, texcoord);

    float luminance =
    0.2989 * source.r +
    0.587  * source.g +
    0.114  * source.b;

    outColour = source;

    if(lutActive)
    {
        outColour = vec4(mix(source.rgb,vec3(luminance, luminance, luminance),uTime), 1);
        vec3 scale = vec3((lutSize - 1.0) / lutSize);
        vec3 offset = vec3(1.0/2.0*lutSize);
        outColour.rgb = texture(customTexure, outColour.rgb * scale + offset).rgb;
    }

}
```

Inside the fragment shader, we use LUTsize to calculate scale and offset for our normal color, then use our 3D texture and normal color to generate the adjusted color as we want. We are alo able to toggle this feature by using the “Q” button in game

Before

After



# Sources


- http://in2gpu.com/2014/06/23/toon-shading-effect-and-simple-contour-detection/
- https://www.defold.com/tutorials/grading/
- https://learnopengl.com/Advanced-Lighting/Deferred-Shading
- https://learnopengl.com/Advanced-Lighting/Bloom